Ground-Up Java

零起点Java应用编程

从零开始学习Java —— 动画演示帮你从新手变成老手

〔美〕 Philip Heller 著

魏海萍 倪 健 黄 玮 译

SYBEX

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

Ground-Up Java

零起点Java应用编程

〔美〕 Philip Heller 著

魏海萍 倪 健 黄 玮 译

電子工業出版社

Publishing House of Electronics Industry

北京 · BEIJING

内 容 简 介

本书是一本从零起点开始介绍Java的书，并以一个图像用户界面的实际开发为例，介绍Java的概念、语句和编程技巧。本书的最大特色是带有30个用于演示Java概念、语句使用和编程技巧的动画插图。这也是迄今为止第一本使用动画插图的图书。

本书是为没有编程经历的人士编写的一本导论性Java程序设计图书，适合初学者、大中专学生和广大编程爱好者阅读。

版权贸易合同登记号　图字：01-2003-7027

图书在版编目（CIP）数据

零起点Java应用编程/（美）海勒（Heller，P.）著；魏海萍等译.—北京：电子工业出版社，2004.4
书名原文：Ground-Up Java
ISBN 7-5053-9693-5

Ⅰ.零…　Ⅱ.①海…　②魏…　Ⅲ.JAVA语言—程序设计　Ⅳ.TP312

中国版本图书馆CIP数据核字（2004）第013276号

责任编辑：春　丽
印　　刷：北京天竺颖华印刷厂
出版发行：电子工业出版社
　　北京市海淀区万寿路173信箱　邮编：100036
　　北京市海淀区翠微东里甲2号　邮编：100036
经　　销：各地新华书店
开　　本：787×1092　1/16　印张：25.25　字数：640 千字
印　　次：2004年4月第1次印刷
定　　价：40.00元

凡购买电子工业出版社的图书，如有缺损问题，请向购买书店调换，若书店售缺，请与本社发行部联系。联系电话：010-68279077。质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。

谨以本书献给Laura

致　谢

首先感谢Denise Santoro Lincoln、Tom Cirtin和Steve Cavin。感谢Michelle、Ricardo和PB&G Productions公司的每个人一直让我能安心写作。永远感谢Simon Roberts、Suzanne Blackstock和Kathy Collina。也感谢Sybex公司的所有高手：Dennis Fitzgerald、Sean Medlock、Kevin Ly、Dan Mummert以及Happenstance Type-O-Rama的Maureen Forys和Jeff Wilson。

简　介

本书是独一无二的，没有哪一本书比得上它，而且它是同类图书中的第一本。知道本书与众不同的原因是很重要的，所以请继续阅读下去。

长久以来，笔者一直认为，编写一本能够由没有编程经历的人阅读的导论性Java程序设计图书是不可能的。这就好像让鱼写关于水的书。人对水了解得不多，但是要把这样的题目介绍给新手非需要这样的了解不可。鱼与水密不可分并天生习惯于水，而与我们这些陆地哺乳动物无法来往，因为我们需要有人把每件事情都解释清楚和分析透彻。一条鱼可能会说“摆动你的尾鳍来向前游，而且不要忘了使用你的鳃”。这对另一条鱼来说是很容易理解的道理，但对你和我是毫无用处的。一本写水的书，即使由海洋中最聪明的鱼来写，仍会充满准确但毫无用处的信息。

Java的情形也是如此。和演奏或吹奏乐器一样，程序设计是一门手艺。而且，和其他任何一门手艺一样，它有它自己的约定、行话和技巧。对于这门手艺的专业人员来说，那些约定、行话和技巧已经成为根深蒂固的习惯、平常的语言和日常生活的各项活动。写一名艺人自己的“栖息地”是非常困难的。

在20世纪70年代，一种叫做C的语言十分流行。在20世纪80年代，C被修改成支持面向对象的程序设计。修改后的C语言叫做C++。这是行话的一个例子。大致说来，在C中，符号“++”意味着“多一点”。因此，C++意味着“C加上多一点”，而且这个含义对任何一名C程序员都是很容易理解的。

20世纪90年代经历了另外一场大演变。C++是一种高效率的语言，但也是一种难学难用的语言。而且，它对最近才发明的新技术，比如高分辨率多彩色显示、数据库或World Wide Web缺少固有的支持。这场新演变的结果叫做Java。这个名称不是一个双关语，也不是任何一个短语的缩写。Java舍弃了C++中已经证明比较麻烦的那些部分，增加了对现代技术的支持。有时，人们把它叫做“C++--++”。这当中有了另外一个符号“-”，它大致表示“少一点”的意思。因此，“C++--++”的意思是“C++加上少一点再加上多一点”。

Java的流行就像夏天的一场大火。C和C++程序设计人员的绝大部分迅速转向了Java，而且看起来永不会回头。为什么那么多的程序员会如此轻易地就选择了这一转换呢？笔者就是他们中的一员。笔者一直靠做C++编程谋生。笔者抽出一年时间写了一部关于一些龙的小说。笔者在完成这部小说之前，用光了所有的钱。幸运的是，那时已是接触Java后的一个月。在几周内，笔者认为自己已是一名有能力的Java程序员，在数月内，笔者就开始教Java和撰写关于Java的文章。荣誉没有光临笔者，但光临了Java的那些设计者们。如果读者了解C和C++，Java是很容易的。如果读者已经会说西班牙语和意大利语，这就像学习葡萄牙语。和同期学习Java的其他人不同，笔者在学习Java所需要的概念、技巧和行话方面已经有了数年的经历。

但是，没有任何程序设计经历的人会怎么样呢？

在笔者学习Java的时候，那时只有两种关于这个题目的图书。现在，有数千种关于这个

题目的书籍（笔者就负责它们中的几种）。除了读者正拿在手中这本书之外，它们当中没有一种是完全从零起点开始介绍程序设计概念的。其他图书多半是准确的，但它们不是有帮助的。

因此，笔者不得不问自己：我能从头开始逐个概念地介绍Java吗？笔者最终意识到，如果能够使用比文字和图片更丰富的某种东西，我就能够做到。是哪种东西使笔者觉得本书是独一无二的呢？这种东西是独一无二的，因为……

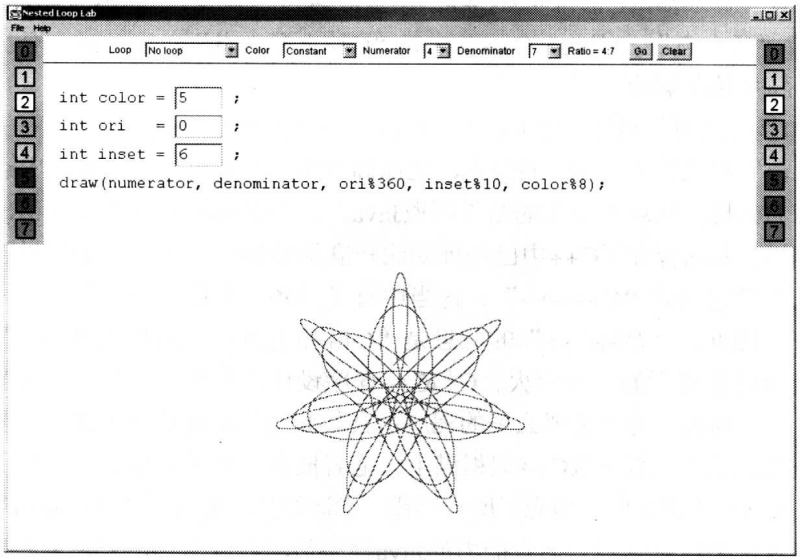
插图是活的

笔者意识到，我真正需要的是一块具有魔力的黑板。

读者可以把一台计算机看做一个巨大的盒子集合，其中每个盒子含有一个数。那些数表示文本、颜色、数据或能够由一个程序所模拟的其他任何东西。那些数以一种复杂的方式随着时间而变化。如果读者只能使用文字和图片，那么描述一个程序的生成周期几乎是不可能的。笔者需要创建会随时间而变化的图片。而且，笔者需要某种超过动画卡通并且在读者每次观看它们时都会相同的东西。笔者需要对读者的好奇心有反应的活插图。笔者需要给读者赋予向那些插图询问“如果……怎么办”问题的威力。

笔者需要某种只能在计算机上使用的东西。这就是本书的动画插图。

本书选配光碟上有30多个动画插图。这些动画是读者可以在自己的计算机上运行的程序。本书给读者提供了关于如何使用它们的完整说明。下面的插图就是一个例子。



这是NestedLoopLab动画的一幅屏幕图（该动画出现在第5章“条件与循环”中）。屏幕中上部分的文本（“int color = 5”等）是Java代码。底部的漩涡状图像是运行该代码的结果。各种控件允许读者改变代码，进而试验不同的值，直到读者熟悉该程序正在做什么时为止。

那些动画插图就像自行车上的练习轮。当读者第一次学骑自行车时，有那么多会出错的事情。如果没有练习轮，读者在摔倒和爬起来上会花费大量时间。练习轮能使读者培养正

确的平衡感。那些动画插图不会允许读者创建发生摔倒的代码。它们提供了一个安全的环境，以便读者能够在里面培养正确的平衡感。

当然，不久以后，该是卸掉练习轮的时候了。在本书每一章的结尾处，读者将会发现一组练习题，它们让读者编写自己的代码。各练习题的推荐解决方法出现在本书的附录部分。

就我所知，本书是使用动画插图的第一本图书。因此，我们不知道它们作为一个教学工具会有怎样的效果。据笔者猜测，它们应该是称职的。每个见过它们的人都非常激动。但是，读者才是最有资格的法官。试一试它们！请让笔者知道你有什么想法。读者可以把自己的建议用电子邮件发给groundupjava@sgsware.com。笔者对知道哪些动画插图最适合你，以及哪些不适合你特别感兴趣。笔者还愿意听到读者可能有的任何建议，以便让更多的动画插图能出现在本书的未来修订版中。动画插图是一项用于学习的新技术，我们欢迎读者参加动画插图开发的行列。

现在……

下载并安装Java

读者在开始编写或运行Java程序以前，需要下载一些软件（那些动画插图是Java程序，因此，如果读者没有做下载，它们将无法运行）。

下载是免费的。在把Java装入到硬盘驱动器上之后，读者必须遵循几个步骤来安装它。这些操作不是很难，但有出错的可能，所以请小心。完整的说明请参见附录A“Java的下载与安装”。

现在，读者随时可以开始。祝你玩得开心！

目　录

第1章 计算机简介

本章内容提要：

- 内存：不完全是0和1
- 内存组织方式
- 简单计算机

Java是一种程序设计语言，用来告诉计算机做些什么。本章将介绍计算机究竟是什么，它们能够做些什么，以及程序员怎样使用程序设计语言来控制它们。

本章将首先剖析计算机处理0和1的秘密。一旦明白了计算机究竟处理些什么之后，我们将介绍它们执行的处理种类。

这绝非一个用脑筋的练习。在这里花些功夫将会使阅读后续章节变得更加轻松。如果读者没有弄懂计算机的基础结构，那么本书后面出现的许多概念，比如数据类型化、引用以及虚拟机将会变得非常难以理解。如果没有理解这些概念，那么学习程序设计就会令人感到困惑和沮丧。但是，如果具备了这些基础知识，那么学习程序设计会是一件令人愉快而有趣的事情。

内存：不完全是0和1

毫无疑问，读者一定听说过计算机只处理0和1。这种说法实际上不完全正确。在选举中，计算机用来计票，所以它们必须能够计数超过1。计算机还用来为质量极其微小的亚原子的行为建立模型，所以除了整数外，它们还必须能够处理小数。它们还用来编写文档，所以除了数字外，它们还必须能够处理文本。

从最底层的结构上看，计算机并不处理0和1、整数、小数或文本。它们是电子电路，所以它们真正处理的仅仅是电流。计算机构件被设计成它们的内部电压要么近似于0伏，要么近似于5或6伏。当一个计算机电路的某一部分达到5或6伏电压时，我们就说它有一个1的值。当一个计算机电路的某一部分达到0伏电压时，我们就说它有一个0的值（幸运的是，这就是读者成为一名高明程序员所需要的全部电学知识）。

因此，翻译就成为了一件十分重要的事情。电压被翻译成0和1。正如读者在本章的较后面和第2章“数据”中将要看到的，0和1被组织成要翻译为数值的组。然后，计算机的较高级部分把那些数值翻译为表示小数、文本、颜色或图像的代码，或者翻译为代表计算机内能够表示的其他无数种对象中的任何一种对象的代码。

现代计算机含有数十亿个在显微镜下才能看得见的微观构件，其中每个构件都有一个0或1的值。我们只关心其内电压近似值的任何一个电路叫做数字电路。用数字电路制造的计算机叫做数字计算机。

注意：数字的对立物是模拟。在模拟电路中，我们关心构件的确切电压。模拟电路对诸如无线电通讯和微波炉之类的某些应用是很理想的，但它们不适用于计算机。模拟计算机曾经在20世纪40年代使用过，但它们是发展过程中的一个死胡同。所有的现代计算机都是数字的。

一种简单但很实用的数字电路类型叫做内存。一个内存电路仅存储一个数字值（0或1，因为程序员不必考虑电压）。内存的一个单独单元叫做一个位（bit），而一个位代表一个“二进制数字”。读者可以将一个位看做是一个微小的盒子，而盒中的内容可以供计算机的其余部分使用。计算机可能会不时地改变盒中的内容。位通常被画成图1.1中所示的样子。

图1.1 一个位

位通常被组织成8位组，叫做字节（Byte）。图1.2给出了一个字节，其中含有0和1的一个任意组合。

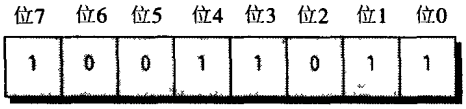


图1.2 一个字节

需要注意的是，那些位从右向左进行编号，而且编号从0开始。计算机设计师总是从0，而不是1开始给事物编号。无论他们给一个字节中的位编号，还是给内存中的字节编号，或者给一个数组中的元素编号，他们都是采用这种方法（正如我们将要在第6章中看到的）。

一个字节可以含有256个位值组合：第0位的两种可能性乘以第1位的两种可能性，再乘以第2位的两种可能性，直至乘以第7位的两种可能性。

如果通过显微镜观看计算机，并看到如图1.2中所示的字节，读者可能很想知道它含有什么值。读者将会看到0和1，但这些0和1有什么含义呢？这是一个没有适合答案的大问题。一个字节可能代表一个整数、一个小数、一个整数或小数的一部分、文档中的一个字符、图像中的一种颜色或程序中的一条指令。这个字节的含义完全取决于它所处的环境。作为一名程序员，读者就是规定每个字节将被怎样解释的那个人。

内存组织方式

一般而言，一台现代个人计算机（PC）含有数亿字节的内存。前缀词“兆”（缩写成M）意指百万，所以我们也可以说一台计算机拥有数百兆字节，即MB。程序和程序员需要一种方法来区别一个字节与另一个字节。这是通过给每个字节分配一个惟一号，叫做字节的地址（Address）来做到的。字节的地址从0开始。图1.3显示了4个字节。

如果图1.3显示512MB，并用同一种比例来绘出，那么它将会高达2000英里（1英里≈1.6公里）左右。

单个字节不是非常有用，因为它的值只限于256种可能性。无论这个字节表示一个数，一个字母，还是计算机应用中的其他任何一种东西，任何东西的256种可能性都不是一个很大的

范围。由于这个缘故，计算机常常使用字节组。联结成一个单元的2个字节可以具有256×256个，即65 536个可能的值。4个字节可以具有256×256×256×256个，即4 294 967 296个可能的值。这就是字节开始变得有用的地方。8个字节可以具有近似于20×10^{18}个不同的值。

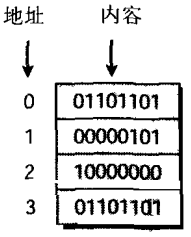


图1.3 几个字节

内存通常以1、2、4或8个字节的块来使用（在下文中，我们将会看到数组或对象使用任意大小的块）。这些块可以表示整数、小数、文本或其他任何一种信息。从这个角度看，我们可以看出“计算机只处理0和1”这句话仅在一个非常有限的意义上是正确的。

读者可以这样理解这句话：一台计算机是一个数字电路，而且我们把它的构件看做具有表示0或1的值。但是，如果注意位于那些数字构件下面的一层，我们将只看到电流，而看不到数字。而且，如果注意位于那些数字构件上面的一层，我们将会看到那些位被组织成由1个或多个字节组成的块，并用来表示多种信息。

除了各种类型的数据之外，内存还可以存储用来操作数据的指令。在下一节中，我们将分析一个非常简单的计算机，并看一看指令与数据是怎样交互的。

简单计算机

本章将介绍一种非常简单的计算机，我们将它称做SimCom。SimCom是一台虚构的计算机。用一个更恰当的词汇来说，它是一台虚拟的计算机。没有人曾经制造过一台SimCom计算机，它只是在本书的选配光碟上的某一个动画插图中被模拟过。

给真实计算机提供动力的各种处理器，如Pentium、SPARC等在质量方面与SimCom没有太大的差别。但是，从质量方面看，存在一个巨大差异：真正的处理器具有更多的指令，更高的速度，以及更大的内存。SimCom却十分简单，它仅仅是一台可以用做一个有用教学工具的计算机。

本节的目标不是让读者成为一名高明的SimCom程序员，而是使用SimCom来介绍程序设计的一些原理。在本书的后续章节中，这些相同的原理将用于Java的上下文中。这些原理包括如下这些：

- 高级语言；
- 循环；
- 引用；
- 2的补码；
- 虚拟机。

在本节中，读者将会了解到一些十分低级的典型处理器元素。像Java这样的现代程序设计语言，有意识地使程序员不必控制这些元素。但是，知道它们存在，以及它们替程序员做什么却是极其有价值的。

SimCom的体系结构非常简单。只有一个32字节的内存存储体；每个字节都可以用来存储一条指令或一个数据。还有一个额外的字节，叫做寄存器（Register），它的作用就像一张草稿纸。另一个叫做程序计数器（Program counter）的构件用来跟踪哪条指令将要得到执行。图1.4显示了SimCom的体系结构。

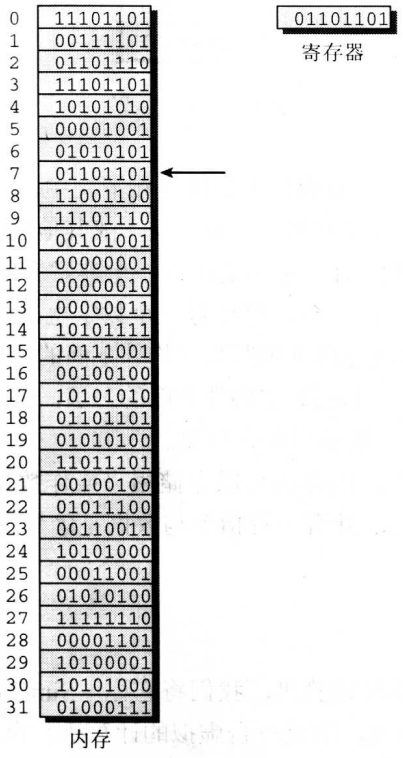


图1.4 SimCom体系结构

图1.4中的箭头代表程序计数器。要执行的下一条指令将是字节7。请注意，字节地址从0开始。

当SimCom启动时，它把程序计数器设置成0。然后，它执行字节0（稍后，我们将会看到这意味着什么）。执行可能会改变寄存器或内存的某一个字节，而且执行几乎总是改变程序计数器。然后，整个过程重复：程序计数器所指出的指令得到执行，并且程序计数器被修改。这将继续下去，直至SimCom收到暂停的指令。

一个指令字节的第7、第6和第5位告诉SimCom做什么。它们叫做操作码（Operation code），也叫做操作码位。第4位至第0位含有辅助指令；它们叫做变元（Argument）位。这部分位如图1.5中所示。

SimCom计算机有7个操作码。表1.1中列出了这7个操作码。

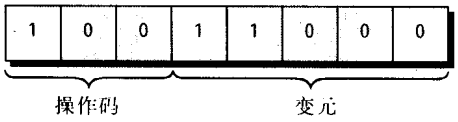


图1.5　操作码和变元位

表1.1　操　作　码

操作码	功能	缩写
000	装入	LOAD
001	存储	STORE
010	加	ADD
011	减	SUB
100	跳转到当前的不同指令处	JUMP
101	寄存器为0时跳转	JUMPZ
110或111	暂停	HALT

这5个变元位含有一个值，并且这个值是一个内存字节的二进制地址。LOAD操作码将这个地址的内容复制到寄存器中。例如，假设当前指令是00000011，那么操作码是000（LOAD），并且变元是00011（3的二进制表示）；当这条指令得到执行时，字节3中的值被复制到寄存器中。需要注意的是，值3没有被复制到寄存器中。变元从不被直接使用；它始终是一个其内容被使用的地址。

STORE操作码将寄存器的内容复制到其地址出现在变元中的那一内存字节中。例如，00100001将使寄存器被复制到字节1中。

ADD操作码将两个值相加。其中的一个值被存储在其地址出现在变元中的那一字节中。另一个值则是寄存器的内容。相加的结果被存储在寄存器中。例如，假设寄存器含有00001100，并且字节1含有00000011，那么指令01000001将使字节1的内容被加到寄存器的内容上，然后相加后的结果被存储到寄存器中。需要注意的是，变元（0001）作为一个地址被间接使用。值0001不直接被加到寄存器上；0001是一个字节的地址，被加到寄存器上的是这个字节的内容。

SUB操作码类似于ADD，所不同的是从寄存器中减去变元所指向的值。相减后的结果被存储在寄存器中。

每当这4个操作码中的一个得到执行之后，程序计数器递增1。因此，控制顺序地流过内存。剩余的3个操作码更改控制的这种正常流程。JUMP操作码不改变寄存器或内存；它只是把它的变元存储在程序计数器中。例如，在执行了10000101之后，要执行的下一条指令将是字节00101（5的二进制表示）中的那条指令。

JUMPZ操作码检查寄存器。如果寄存器含有00000000，程序计数器就被设置成指令的变元。否则，程序计数器仅被递增（即递增1），并且控制正常流动。这是一个非常有威力的操作码，因为它能使计算机感知它的数据，并根据不同的条件做出不同的反应。

最后，HALT操作码致使计算机停止处理。

现在，让我们来看一个简短的程序。

```
00000100
01000100
00100100
11000000
```

关于这个程序，可以注意到的第一件事情是它阅读起来很困难。下面，我们将它翻译成一种更为友好的格式：

```
LOAD 4
ADD 4
STORE 4
HALT
```

这个程序使字节4中的值加倍。它是通过把该值复制到寄存器中，然后把这个相同的值加到寄存器上，最后把相加后的结果再存储到字节4中来完成这一计算的。

本例说明，通过操纵0和1来进行编程不是什么好事情。这些含义清楚的操作码和十进制数，是高度结构化和差别细微的人类语言与计算机二进制语言之间的一种折中。LOAD 4表示方式叫做汇编语言（Assembly language）。在汇编语言中，一行代码一般对应于一条单独的计算机指令，并且计算机必须始终知道计算机的体系结构和状态。汇编程序（Assembler）是指将汇编语言翻译成二进制表示的一种程序。

试验SimCom

令人遗憾的是，我们无法给本书的每份拷贝都包装一个SimCom，但我们已经做了下面一件最好的事情。本书选配光碟上的第一个动画插图就是SimCom的一个操作性模拟。

注意： 如果读者还没有在自己的计算机上安装Java，现在该是安装它的时候了。如果不十分清楚怎样安装Java，请参考本书的附录A“Java的下载与安装”，该附录将指导读者完成整个下载和安装过程。在本书中，我们将邀请读者一起来运行动画插图，而且将给读者提供要键入到计算机上的命令。通读一遍附录A是十分有必要的。

要想运行这个SimCom模拟，在命令提示符处键入下列命令：

```
java simcom.SimComFrame
```

这个模拟允许读者装入并执行预先已存在的程序或创建新的程序。图1.6显示了这个模拟在操作中的情形。

内存中的每个字节以3种格式显示：二进制、十进制以及操作码加变元。寄存器仅以二进制和十进制格式显示；由于寄存器从不被执行，所以显示哪条指令将要得到执行没有任何意义。首先，通过单击内存中的任一字节的内部，读者可以改变该字节。其次，如果单击十进制区域，将会获得一个面板，该面板允许读者选择一个新的十进制值。如果单击操作码区域，将会获得一个允许选择新操作码的面板。要想更改变元，先单击选定字节的变元区域。随着读者移动鼠标，最接近的字节地址将突出显示出来。当需要的地址突出显示时，单击它就可以将它设置为变元。

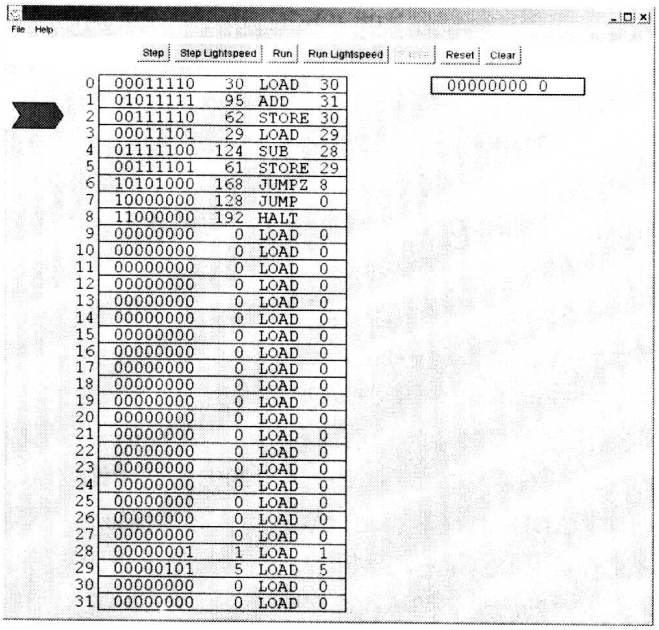


图1.6　操作中的SimCom

试着执行一个十分简单的程序。在File（文件）菜单中单击File➤Scenarios，然后选择Load/Add/Store/。这个程序将字节10和字节11（不是数值10和11）相加，并把相加后的结果存储在字节12中。最初，字节10和字节11都为0，所以要想看到有趣的结果，读者就必须改变它们的值。要想了解这个程序在操作中的情形，单击Step按钮。这将以慢动作的形式执行当前指令。要想连续运行，单击Run按钮，这将连续播放本动画，直至执行一条HALT指令。如果厌烦了慢动作，则可以单击Step Lightspeed或Run Lightspeed按钮来立即获得结果。Reset按钮重新初始化内存，并把程序计数器设置成0。

试一试在字节10和字节11中存储相当大的值。一个字节能够存储的最大值是255。如果试一试相加5+255，会出现什么情形呢？

更改这个程序，使字节10减去字节11。如果字节10含有5，而字节11含有6，会出现什么情形呢？

当准备好了一个更有趣的程序时，单击File菜单中的Scenarios➤Times 5。这个程序将字节31的内容乘以5，并将运算结果存储在字节30中。在字节31中试验一个新值来确信它工作。在每次运行之后，请不要忘了单击Reset按钮。

这个程序看起来复杂得可能有些多余。令人遗憾的是，SimCom指令集没有包含乘法操作码，但是，由于它没有包含乘法操作码，所以下列程序更直截了当，不是吗？

```
LOAD 31
ADD 31
ADD 31
ADD 31
ADD 31
STORE 30
HALT
```